Documentation

Overall summary of your program

An important thing to understand is that instruments and vocals that sound great on their own can start to sound very messy when mixed with other instruments and vocals. Many inexperienced audio producers, audio engineers, and vocalists find it difficult to master audio when presented with this issue. Hence, we created MUSE.

MUSE is an object oriented C++ Virtual Studio Technology (VST)—an audio Equalizer (EQ) plug-in software and interface—that utilizes the JUCE framework in order to shift and filter frequencies to a user's desired preferefence in order to create clean mixes.

Inputs & Outputs

In order to process the audio, the source of the audio gets placed into an input. After the input audio runs through the MUSE EQ filter, the audio is then processed and altered. After the audio meets the user's preference it is then outputted to the host Digital Audio Workstation (DAW)—software used for recording, editing and producing audio files—and the changes can be heard by the user.

Relationship between the files

A standard JUCE VST extension software works from only four "source" files; however, these source files are capable of referencing functions and data types present within all of the JUCE libraries. The templates for these files: PluginEditor.h, PluginEditor.cpp, PluginProcessor.h, and PluginProcessor.cpp are generated when creating a new JUCE VST extension project, but serve only as a broad outline for the program, which needs extensive alteration and modification to serve its main purpose, to provide a user interface for shifting and filtering frequencies for their audio.

PluginProcessor.h and PluginProcessor.cpp are the files responsible for the audio manipulation that MUSE offers to its user. It is these files which specify the input values and their functions as they relate to the altering of the users desired audio. PluginEditor.h and PluginEditor.cpp are responsible for the creation of the GUI, which focuses on creating elements on the display via sectioning off parts of the windowed display for seperate objects. Each of these display themselves in their given region.

Relationship between the classes, data structures, etc.

The AudioProcessorValueTreeState (APVTS) is a class that is linked to a value tree that identifies values by a string ID. This class provides a baseline to help link the GUI to sliders and also allows you to provide a parameter layout. After creating a parameter layout, you can use this layout to create an APVTS object. Using the Filter class originating from the IIR and DSP classes (Digital Signal Processing) we were able to take the parameters we made and allow them to manipulate audio. We created Filters (Peaks) and CutFilters (High + LowCut) and then organized them how we wanted by using the DSP processor chain class (in the order of CutFilter, Peak1, Peak2, Peak3, CutFilter). After doing this, we got the settings for the chain that we made when doing the layout by passing the APVTS to a function called getChainSettings and converting some of the base values into raw values that the program could understand. Then after

all this is done, there is a class called Coefficients that creates the base for making a filter based off of the settings (Frequency range, gain range and quality range) we had from the parameter layout. A function called updatePeakFilters was then made which updates the peakCoefficients and allows the signal to be changed based on what the user chooses for the value. We repeated the chain settings, and Coefficient settings for the CutFilters after. Once we had all of these made, we created a function to update the cutFilters and pass all of the update functions to the processBlock function so that as audio gets processed the filters get changed—if the users decide to change them or let the filters stay the same. The Process block then processes the signals in the left and right audio channel and sends it back to the output of the program (typically the DAW).

Interesting core algorithms

An interesting aspect of the code is the way in which the GUI is created through the code in the PluginEditor files. The entire window visual is described in the GUI as the FiveBandEQAudioProcessorEditor. This class derives from the JUCE AudioProcessorEditor class, which enables all of the unique JUCE functions from the libraries for creating visuals.

The interesting part about this class and how it works is that the visuals displayed to the user are all instances of objects within the main object. The benefit of this is that an object, say a rotary slider that needs to be displayed on the screen to the user, can be instantiated as many times as one wants in the main function, so only one "slider" object needs to be designed. The main object paints the main background of the display and also creates the bounds for which objects within itself can be painted on.

This RotarySliderWithLabels object receives its ability to manipulate the data through a parameter value which associates it with a slider value in the PluginProcessor files. This makes it so that user input into the knobs on the screen actually alters the audio being input into it.

The ResponseCurveComponent is another object instantiated when FiveBandEQAudioProcessorEditor is constructed. This component does three things, it checks for any changes in the slider parameters, then, if there are changes in them, it updates the values that the response curve is going to show, it then paints itself onto the screen in a region specified in the resized function within FiveBandEQAudioProcessorEditor.

The truly interesting part about the GUI created through these files is the way in which one object creates the layout of the window and constructs other visual objects which are able to populate the fields specified using their own paint functions and the JUCE library.

Functions of interest

Within the PluginEditor files, three functions of interest are resize, paint, and the constructor for FiveBandEQAudioProcessorEditor.

The resize function is overridden from the base meaning of the function in the JUCE libraries for each object within the GUI code. Resize establishes the regions of each section that are dedicated to which objects, and the paint function is called for every object allowing it to draw itself in the region granted to it by the resized function. It is interesting how all the functions work together to display the components.

The constructor for FiveBandEQAudioProcessorEditor is interesting because it shows how each component is instantiated in the code, showing how use of the slider class to create multiple sliders in one class made it easier to implement module display. This was because a for loop could be used to iterate through each instantiation in the FivaBandEQAudioProcessorEditor class and use the same functions with different parameters to display the objects.

Challenges

One of the key challenges of this project was figuring out how to utilize the JUCE framework on our coding editors. Each group member had different laptops and utilized different coding editors. While each team member was able to download the JUCE framework from github successfully, the problem lied when trying to run it on their preferred text editor. For example, when trying to run the framework on VS code, some of the group members ran into an issue because of why it can not build .sln files on the other hand Visual Studio was capable of doing that so it was a simple switch to the other IDE. Also, JUCE is more suitable on Xcode, a software only on an Apple environment, which not every team member had. While one of our team members was able to successfully use Xcode and run the JUCE framework, everyone else was obligated to rely on Visual Studio to run JUCE.

Another key challenge with this project was actually working on it. Due to conflicting and overloaded schedules, it was difficult to find times to meet together and work on the project. However, this was easily solved by making an agenda for each team member that they could stick to and follow.

Why this was a good project and deserves 15% of your grade

Unlike any other software project, this group project allowed everyone to become familiar with open source software from Github - a real world software engineering skill. Using what we learned from the lab, we were able to clone the repository onto our computers and push our commits to our repo. This allowed us to collaboratively work on code and fix any errors one code could not solve but another team member could. As opposed to programming a game, our group wanted to do something different and create something useful to other people, more specifically audio engineers and vocalists. In order to use C++, we quickly familiarize ourselves with JUCE, a C++ framework used for audio plug-ins. Most importantly, we were able to utilize many of the topics learned throughout the semester into this project: object oriented programming, software design, debugging, compilation, and program management. An interesting thing about this project is the fact that we as a group are not done. While this class may end, in the next week, we plan to add to our plug-in, making more enhancements and hopefully making it public on our repositories. Overall, this project was a great way to experience software engineering before actually becoming a software engineer.